Database indexing | Hash Indexes

Suresh Podeti

3 min read · May 14, 2023

Photo by Matt Artz on Unsplash

In my previous article I have explained how database storage and retrieval works with an example:

World's Simplest database | Using bash functions

In this article I will explain how database storage and retrieval works with an example.

medium.com

In this article I will explain how database indexes works with special focus on hash based indexes. These indexes are not the only kind of data you can index, but it's very common, and it's a useful building block for more complex indexes.

An index is an **additional structure** that is **derived from the primary data.** Indexes doesn't affect the contents of the database; it only **affects the performance of queries.** Maintaining additional structures **incurs overhead**, especially on **writes.** Any kind of index usually slows down writes, because the **index also needs to be updated every time** data is written. So, bottom-line is **well-chosen indexes speeds up read queries, but every index slows down writes.** For this reason, databases don't usually index everything by default.

An additional structure for index can be hash map, also called dictionary in some programming languages. So, **what we need to store in index hash map?.** We need to store mapping of key to byte offset in the data file — the location at which the value can be found, as illustrated in Figure 1.

Whenever you append a new key-value pair to the file, you **also update the hash map** to reflect the offset of the data you just wrote (this works both for inserting new keys and for updating existing keys).

When you **want to look up a value**, **use the hash map** to find the offset in the data file, seek to that location, and read the value.

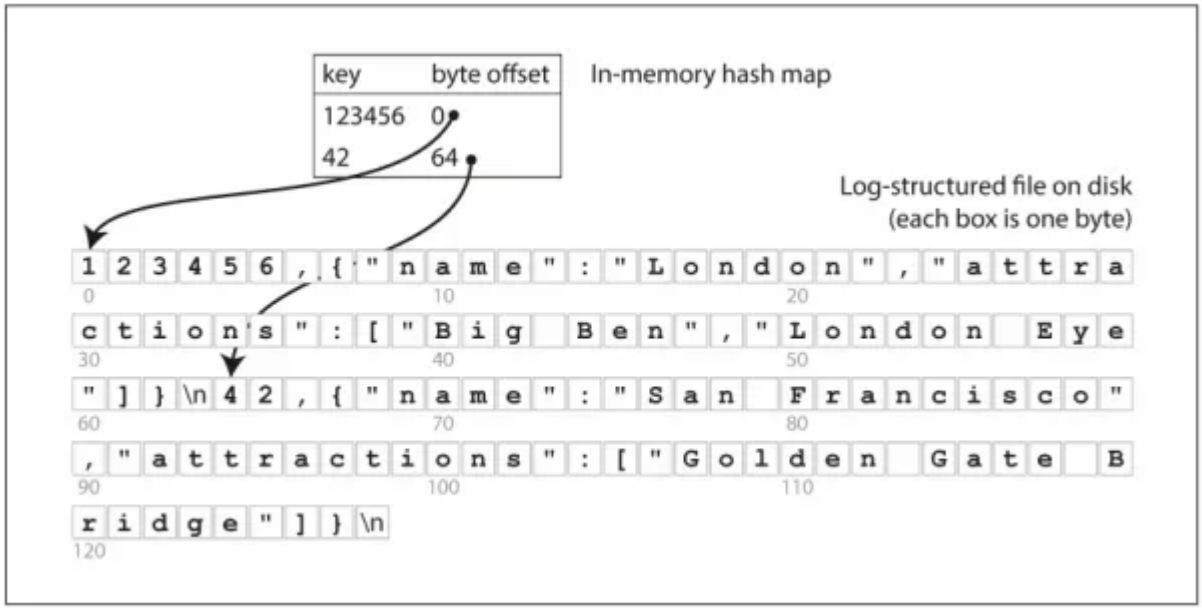

Figure 1. Storing a log of key-value pairs in a CSV-like format, indexed with an in-memory hash map

If the **database is restarted**, the in-memory **hash maps are lost.** We can re-construct has by reading the log and constructing indexes from scratch but that might take a long time if the log file is large. So, server restarts are painful. One of the possible solution could be is storing the snapshot of hash map on disk, which can be loaded into memory more quickly.

Hash based database indexing may sound simplistic, but it is a viable approach. In fact, this is essentially used in Bitcask (the default storage engine in **Riak**).

However, the hash table index also has limitations:

1. The **hash table must fit in memory**, so if you have a very large number of keys you're out of luck. In principle, you could maintain a hash map on disk, but unfortunately it is difficult to make an on-disk hash map perform well. It requires a lot of random access I/O.
2. **Range queries are not efficient.** For example, you cannot easily scan over all keys between kitty00000 and kitty 99999 — you'd have to look up each key individually in the hash maps.

In the next article, I will look explain an indexing structure that doesn't have those limitations.

References:

1. O'Reilly designing data-intensive applications by Martin Kleppmann, Chapter 3: Storage and Retrieval

Database | Database Indexing | Hash Index | Log Files | Key Value Store

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium